Python Programming

Principal Component Analysis

Dr. Chun-Hsiang Chan
Department of Geography
National Taiwan Normal University

Outlines

- Review
- Why Do We Need Dimension Reduction?
- PCA – Geometry
- PCA – Linear Algebra
- PCA – Characteristics
- PCA – Principal Components
- PCA – Variable Loading
- PCA – Steps of PCA
- PCA – sklearn. decomposition.pca
- PCA – SVD
- PCA – Eigen Decomposition
- PCA– Variable Loading

Review

- Before we explain PCA, we need to review the mathematical meaning of three basic descriptive statistics, including expectation, variance, and covariance.
- In previous courses or your understanding, these three parameters usually perform as above equations.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

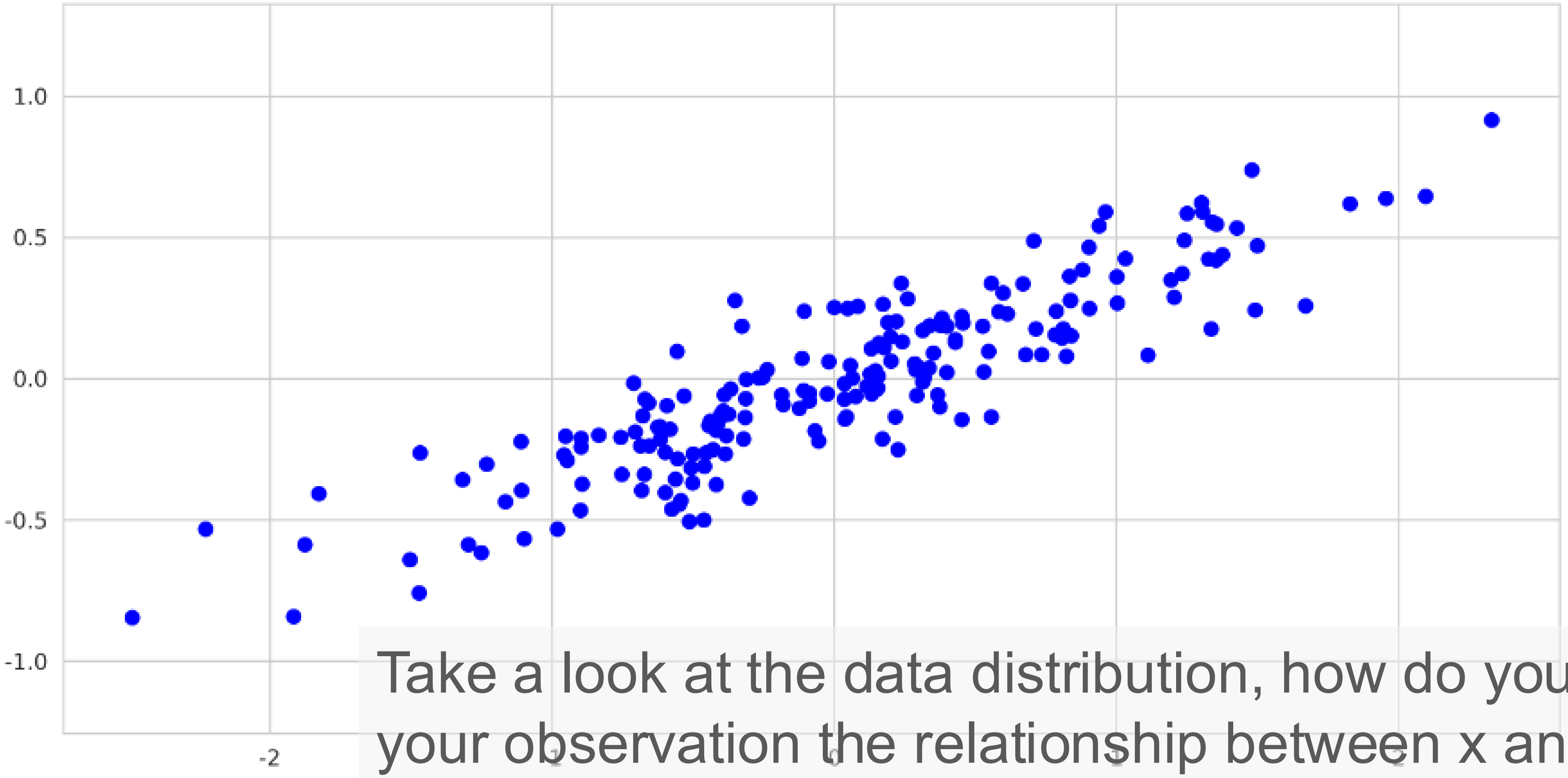
$$var(x) = \sigma^2 = \left(\sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$Cov(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

Review

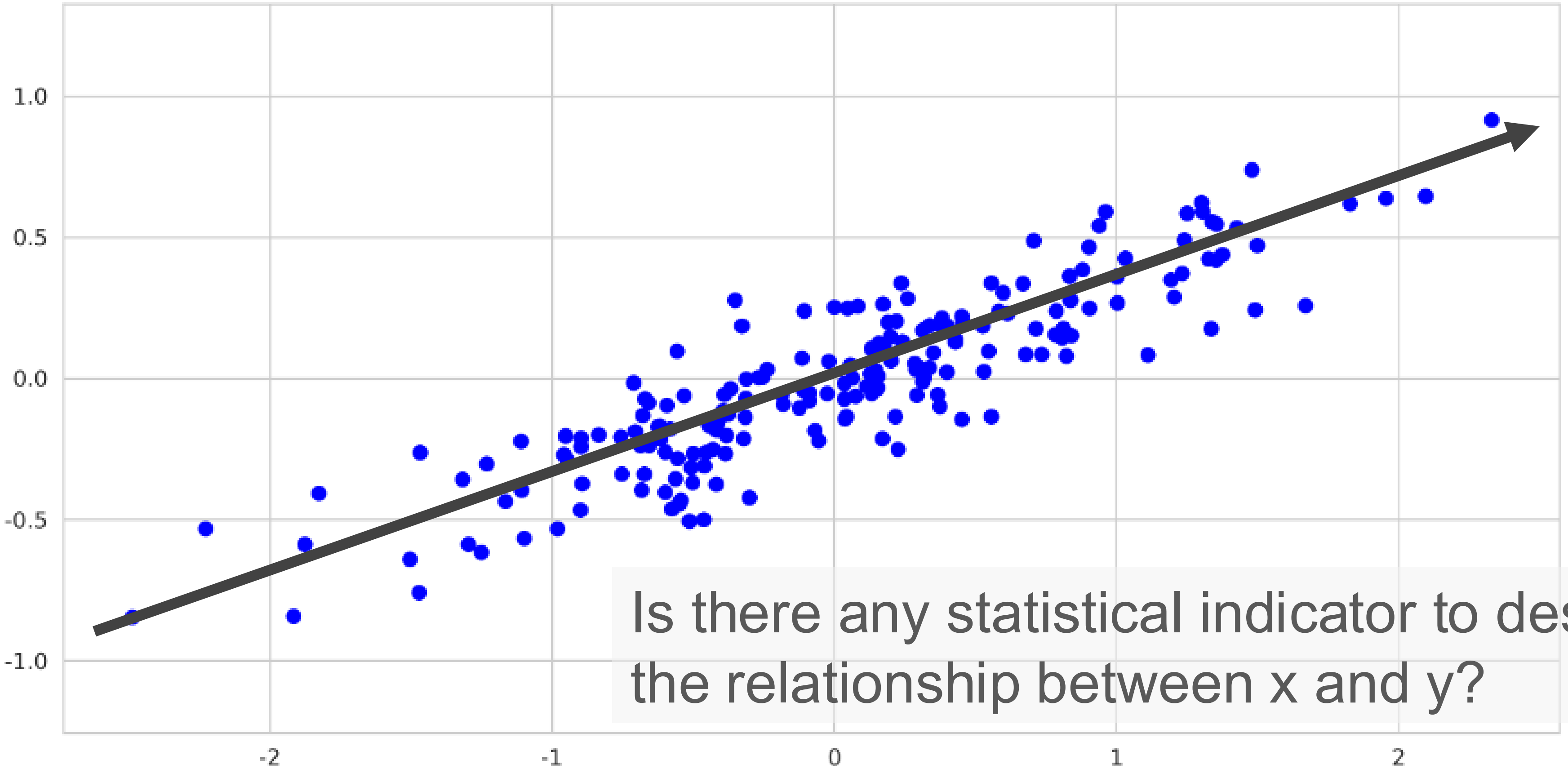
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid\ var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$



Take a look at the data distribution, how do you explain your observation the relationship between x and y?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \mid var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)$$

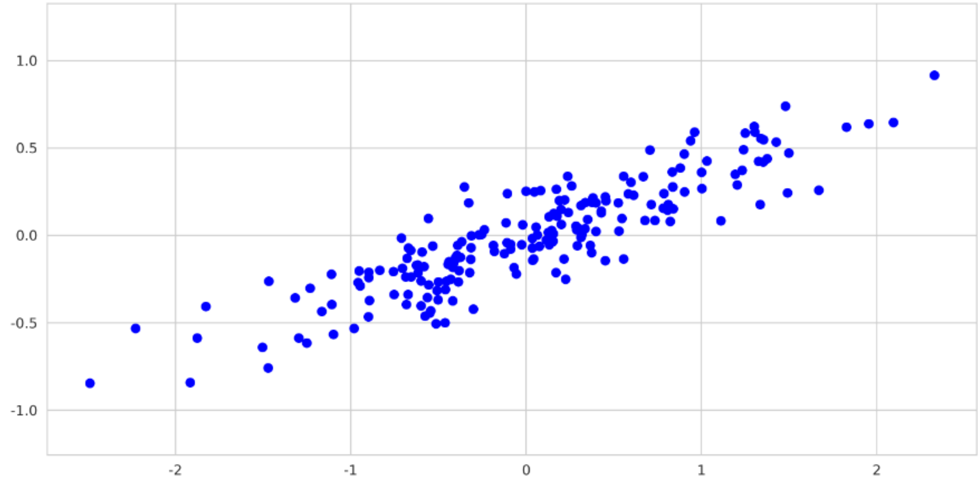


Is there any statistical indicator to describe the relationship between x and y?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \mid var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)$$

- Pearson‘s correlation coefficient
- Given two parameters x_i and y_i, where i ranges from 1 to n. Then, Pearson’s correlation coefficient could be defined as follows.



$$\rho = \frac{\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i-\mu_x)^2 \sum_{i=1}^{n}(y_i-\mu_y)^2}}$$

Question 1

If x is highly correlated with y, and then what do you expect from their covariance and standard deviations?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid \ var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \mid Cov(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

- **Expectation**

$$E(X) = \sum_{x} xP(X = x) = \mu$$

- **Variance**

$$\begin{aligned} var(X) &= E([X-\mu]^2) \\ &= E(X^2 - 2\mu X + \mu^2) \\ &= E(X^2) - 2\mu E(X) + \mu^2 \\ &= E(X^2) - 2\mu^2 + \mu^2 \\ &= E(X^2) - \mu^2 \\ &= E(X^2) - E(X)^2 \end{aligned}$$

Characteristics of Expectation

$E(aX + bY) = aE(X) + bE(Y), a, b \in \mathbb{R}$

X and Y can be independent or dependent.

$E(XY) = E(X)E(Y)$

Where $cov(X, Y) = 0$

Review

$E(aX+bY)=aE(X)+bE(Y), \mathrm{a,b}\in\mathbb{R}$
X and Y can be independent or dependent.
$E(XY)=E(X)E(Y)$
Where $cov(X,Y)=0$

Covariance

- If x and y are independent…

$$var(X+Y)=var(X)+var(Y)$$

- If x and y are dependent…

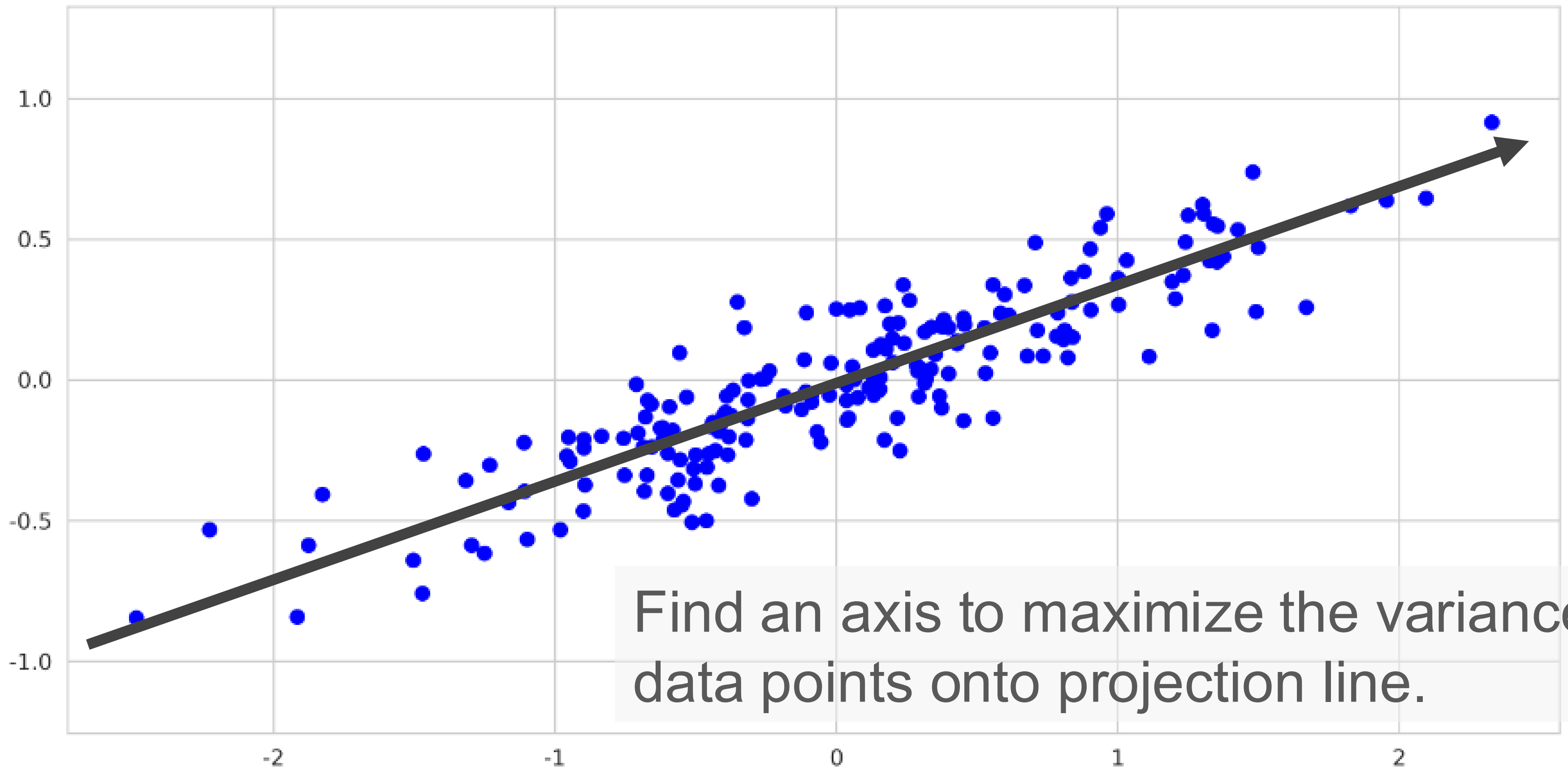
$$var(X+Y)=E([(X+Y)-E(X+Y)]^2)$$
$$=E\left(\left[(X+Y)-\left(E(X)+E(Y)\right)\right]^2\right)$$
$$=E\left(\left[\left(X-E(X)\right)+\left(Y-E(Y)\right)\right]^2\right)$$
$$=E\left(\left(X-E(X)\right)^2+2\left(X-E(X)\right)\left(Y-E(Y)\right)+\left(Y-E(Y)\right)^2\right)$$
$$=E\left[\left(X-E(X)\right)^2\right]+E\left[\left(Y-E(Y)\right)^2\right]+2E[\left(X-E(X)\right)\left(Y-E(Y)\right)]$$
$$=var(X)+var(Y)+2cov(X,Y)$$

Why Do We Need Dimension Reduction?

- Here comes the first question into your mind.
  - Why do we need dimension reduction?
  - What's the importance of dimension reduction?
  - Can we directly import all datasets into your model without dimension reduction?
- Statistical models (e.g., linear regression) have several assumptions when you adopt them. One of them is "all variables have to be linearly independent," indicating no collinearity.
- To achieve this goal, various methods were developed for orthogonalizing parameters and reducing the dimension of the dataset, such as PCA, LDA, LLE, and Laplacian Eigenmaps.

PCA – Geometry

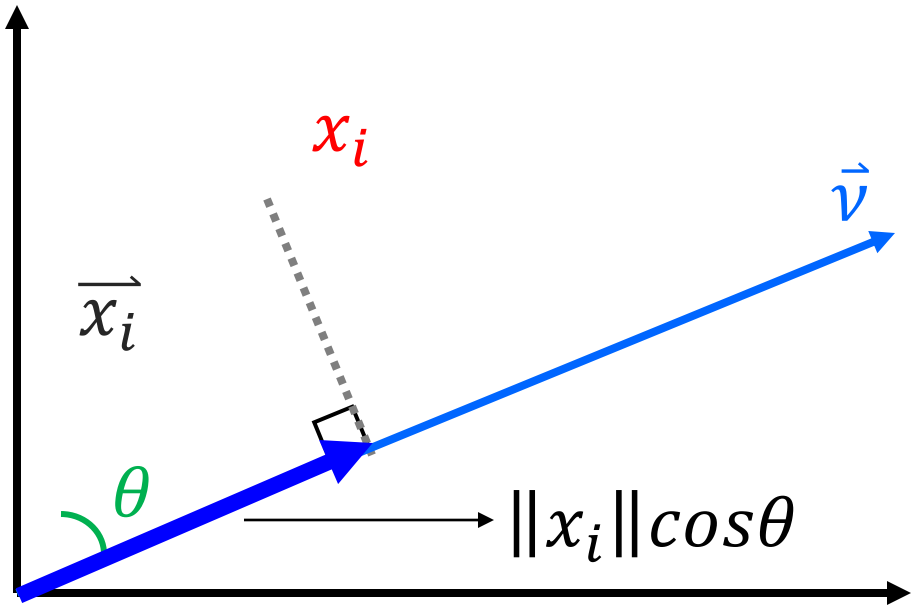
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid\ var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)$$



Find an axis to maximize the variance of all data points onto projection line.

PCA – Geometry

- Given a point "x" and project onto a vector "v".



$$cos\theta = \frac{{x_i}^T \cdot v}{\|x_i\|\|v\|}$$

$$\|x_i\|cos\theta = \|x_i\| \frac{{x_i}^T \cdot v}{\|x_i\|\|v\|} = \frac{{x_i}^T \cdot v}{\|v\|}$$

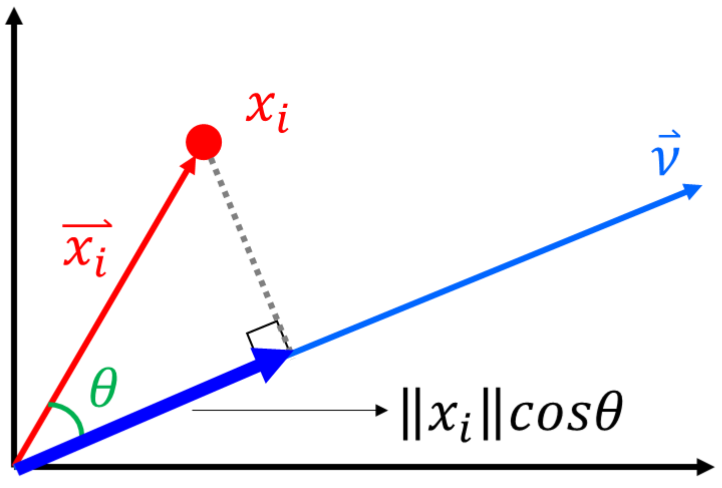
$$if\ v\ is\ unit\ vector\ \ldots\ \|v\| = 1$$

$$= \frac{{x_i}^T \cdot v}{\|v\|} = {x_i}^T \cdot v$$

PCA – Linear Algebra

$$X = \begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \rightarrow X^T = \begin{bmatrix} - & x_1 & - \\ - & x_2 & - \\ - & \vdots & - \\ - & x_n & - \end{bmatrix}$$

$$P = \begin{bmatrix} {x_1}^T \\ {x_2}^T \\ \vdots \\ {x_n}^T \end{bmatrix} = X^T u \Rightarrow solve\ P$$



$${x_i}^T \cdot v$$

PCA – Linear Algebra

$$J(u) = \|P^2\| = P^T P = (X^T u)^T (X^T u) = u^T X X^T u$$

$$\underset{u}{\operatorname{argmax}}\ J(u) = u^T X X^T u, where\ subject\ to\ u^T u = 1$$

Add Lagrange multiplier

$$\underset{u,\lambda}{\operatorname{argmax}}\ J(u,\lambda) = u^T X X^T u + \lambda(1 - u^T u)$$

$$\nabla_u J(u,\lambda) = \nabla_u\left(u^T X X^T u + \lambda(1 - u^T u)\right) = 0$$

$$\Rightarrow 2XX^T u - 2\lambda u = 0$$

$$\Rightarrow \underbrace{\boxed{XX^T}}_{cov(X)} u = \underbrace{\boxed{\lambda}}_{\text{eigenvalue}}\boxed{u} \rightarrow \text{eigenvector} \longrightarrow Au = \lambda u$$

PCA – Linear Algebra

$When\ u\ is\ the\ eigenvector$

$$J(u) = \|P^2\| = u^T XX^T u = u^T \lambda u = \lambda u^T u = \lambda$$

$$XX^T u = \lambda u$$

Given an ***eigenvector****, the* ***total square of projected values*** *is the* ***eigenvalue*** *=* λ

An eigenvector is a symmetry matrix

u is n unit vector

$$uu^T = u^T u = 1$$

PCA – Characteristics

$$XX^T u = \lambda u$$

An eigenvector is a symmetry matrix
u is a unit vector

$$uu^T = u^T u = 1$$

A is a square symmetric matrix has orthogonal eigenvectors with different eigenvalues.

$[x_1, \lambda_1]$, $[x_2, \lambda_2]$

$$\begin{cases} Ax_1 = \lambda_1 x_1 \\ Ax_2 = \lambda_2 x_2 \end{cases}$$

Equal

$$x_1{}^T A x_2 = x_1{}^T \lambda_2 x_2 = \lambda_2 x_1{}^T x_2$$

$$x_1{}^T A^T x_2 = (Ax_1)^T x_2 = (\lambda_1 x_1)^T x_2 = \lambda_1 x_1{}^T x_2$$

$$(\because A \in symmetric\ matrix, \therefore A = A^T)$$

$$\lambda_2 x_1{}^T x_2 = \lambda_1 x_1{}^T x_2$$

$$\underbrace{x_1{}^T x_2}_{\text{Orthogonal } x_1{}^T x_2 = 0}\underbrace{(\lambda_2 - \lambda_1)}_{\text{All eigenvalues are different}} = 0$$

PCA – Characteristics

- Conversion between orthogonal bases

$$u_i \cdot u_j = u_i^T \cdot u_j = \begin{cases} 1, if\ i = j \\ 0, otherwise \end{cases}$$

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_d \\ | & | & & | \end{bmatrix} \Rightarrow U^T U = I = U^{-1} = U^T$$

$$x = y_1 u_1 + y_2 u_2 + \cdots + y_d u_d = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_d \\ | & | & & | \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} = Uy$$

$$\Rightarrow y = U^{-1} x = U^T x$$

PCA – Princial Components

Principal components ($\mathbf{PC}$)

- Centered Matrix ($A^{N\times p}$ with N samples and p features)

$$A_{centered} = A - \bar{A}$$

- Compute the covariance matrix

$$S = \frac{1}{N} A_{centered}^{T} A_{centered}, S \in p \times p\ matrix$$

- Apply Eigen Decomposition

$$\boldsymbol{SV = \lambda V}$$

V (eigenvectors) define **principal component directions**
λ (eigenvalues) define **variance explained** by **each** $\boldsymbol{PC}$

PCA – Principal Components

- The eigenvectors of S are the Principal Components (PCs)

$$PC = A_{centered}V$$

- Using Singular Value Decomposition (SVD)

$$A_{centered} = U\Sigma V^T$$

U contains left singular vectors
Σ contains singular values
V contains right singular vectors (principal component directions)

$PC \Rightarrow A_{centered}V = U\Sigma V^T V$
$\because V^T V = I, \therefore \boldsymbol{A_{centered}V = U\Sigma = PC}$
*Each **row** of PC is a transformed data point*
***Columns** of PC correspond to principal compornents.*

PCA – Principal Components

- **Compute PCA to Covariance Matrix**
- PCA is based on the Eigen Decomposition of the covariance matrix

$$S = \frac{1}{N} A^T A$$

- Using Singular Value Decomposition (SVD)

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T$$

- Since U is orthogonal, $U^T U = I$

$$A^T A = V\Sigma^T \Sigma V^T$$

- Divide by N

$$\frac{1}{N} A^T A = \frac{1}{N} V\Sigma^T \Sigma V^T$$

PCA – Principal Components

$$\frac{1}{N} A^T A = \frac{1}{N} V \Sigma^T \Sigma V^T = V \frac{\Sigma^T \Sigma}{N} V^T$$

- From Eigen Decomposition, the **eigenvalues** of S are:

$$E = \frac{\Sigma^T \Sigma}{N}$$

- Taking the square root:

$$\Sigma = \sqrt{N}\sqrt{E}$$

PCA – Variable Loading

- From the covariance matrix, we can obtain variable loading

$$cov(A, PC) = \frac{1}{N} A^T PC, \because cov(X, Y) = \mathbb{E}\left[(X - \mu_x)\left(Y - \mu_y\right)\right]$$

- Subsitution with $\boldsymbol{PC = U\Sigma = A_{centered}V}$

$$cov(A, PC) = \frac{1}{N} A^T U\Sigma, where\ \Sigma = \sqrt{NE}$$

- Since E is just a scaling factor; therefore, we may approximate

$$cov(A, PC) = \frac{1}{N} A^T U\sqrt{N} = \frac{\sqrt{N}}{N} A^T U = \frac{1}{\sqrt{N}} A^T U$$

PCA – Variable Loading

$$cov(A, PC) = \frac{1}{\sqrt{N}} A^T U$$

- Since Singular Value Decomposition (SVD)

$$A^T = (U\Sigma V^T)^T = V\Sigma^T U^T$$

$$cov(A, PC) = \frac{1}{\sqrt{N}} V\Sigma^T U^T U$$

- Since U is an orthogonal matrix, $\therefore U^T U = I$, and Σ is a diagonal matrix with singular values, $\therefore \Sigma^T = \Sigma, \text{ and } \Sigma = \sqrt{NE}$

$$cov(A, PC) = \frac{1}{\sqrt{N}} V\Sigma = V \frac{1}{\sqrt{N}} \sqrt{NE} = V\sqrt{E}$$

PCA – Variable Loading

- Finally, the variable loading could be obtained

$$cov(A, PC) = \mathcal{L} = V\sqrt{E}$$

  $\mathcal{L}$ is the variable loading of each PC

  V is the eigenvector of A

  E is the eigenvalues of A

PCA – Steps of PCA

The Steps of PCA

1. Find the sample mean $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
2. Subtract mean
3. Compute covariance matrix $C = \frac{1}{n}XX^T = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)\,(x_i - \mu)$
4. Find the eigenvalues of C and arrange them into descending order
   $\lambda_1 > \lambda_2 > \cdots > \lambda_d, \{u_1, u_2, \dots, u_d\}$
5. The transformation is $y = U^T X$.

PCA – Python Tutorial

- As mentioned earlier, Principal Component Analysis (PCA) can be computed using different methods, each with its own advantages:
  1) **Pre-built Libraries:**

     The simplest approach is using `sklearn.decomposition.PCA`, which internally applies Singular Value Decomposition (SVD) for numerical stability and efficiency.

PCA – Python Tutorial

2) **Eigen Decomposition**:

   This method computes eigenvalues and eigenvectors of the covariance matrix ($A^T A/N$), where eigenvectors define the principal component directions and eigenvalues indicate the explained variance. It works best when the covariance matrix is well-conditioned.

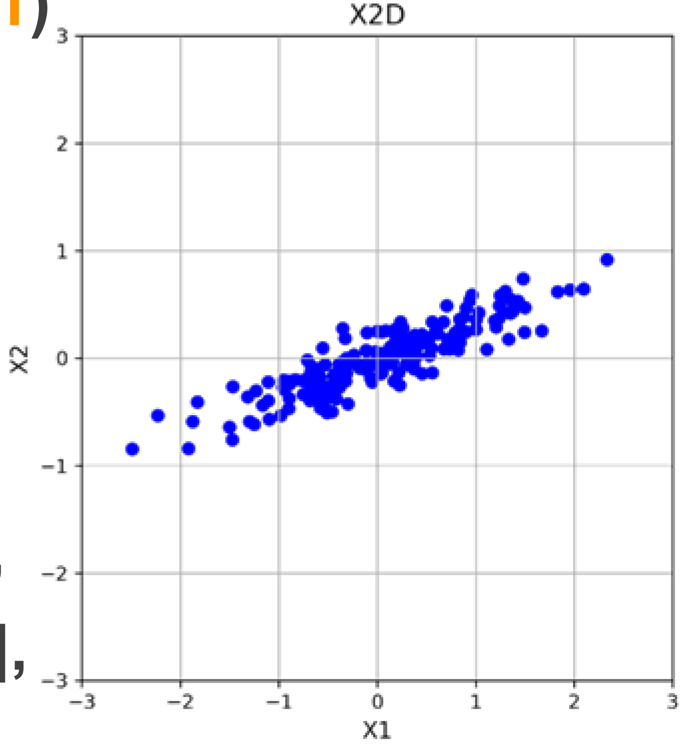
3) **Singular Value Decomposition (SVD)**:

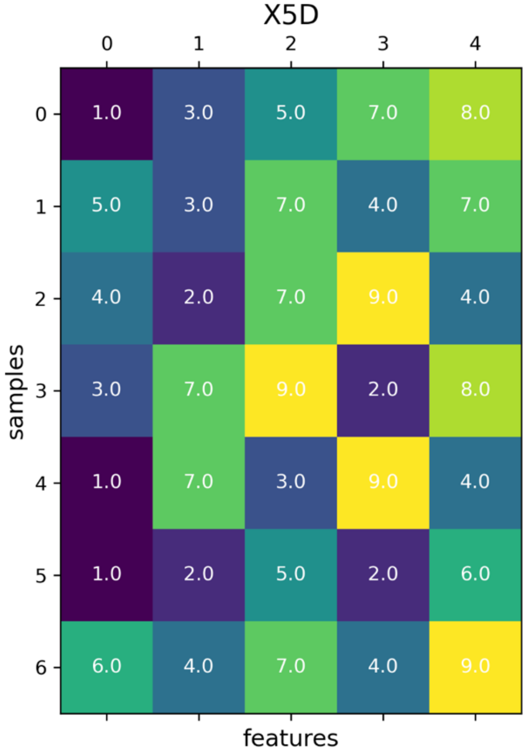
   Instead of explicitly computing the covariance matrix, SVD decomposes the mean-centered data matrix as $A = U\Sigma V^T$.

- While eigen decomposition offers an intuitive understanding, SVD is generally preferred for its robustness and efficiency.

PCA – sklearn.decomposition.pca

```
# define a m a trix with 200 samples with two features
rn_state = np.random.RandomState(1)
# 2x200 -> 200x2
A2D = np.dot(rn_state.rand(2, 2),
          rn_state.randn(2, 200)).T
# define a matrix with seven samples
# and five features
A5D = np.array([[1,3,5,7,8],[5,3,7,4,7],
          [4,2,7,9,4],[3,7,9,2,8], [1,7,3,9,4],
          [1,2,5,2,6],[6,4,7,4,9]])
```





PCA – sklearn.decomposition.pca

```
# set the number of components for PCA model
pca2D = PCA(n_components=2)
pca5D = PCA(n_components=5)
```

```
# fit PCA model
pca2D.fit(A2D)
```

```
▾ PCA
PCA(n_components=2)
```

```
pca5D.fit(A5D)
```

```
▾ PCA
PCA(n_components=5)
```

PCA – sklearn.decomposition.pca

```
# principal component - eigenvector
# (n_components, n_features)
print(pca2D.components_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

```
print(pca5D.components_)
```

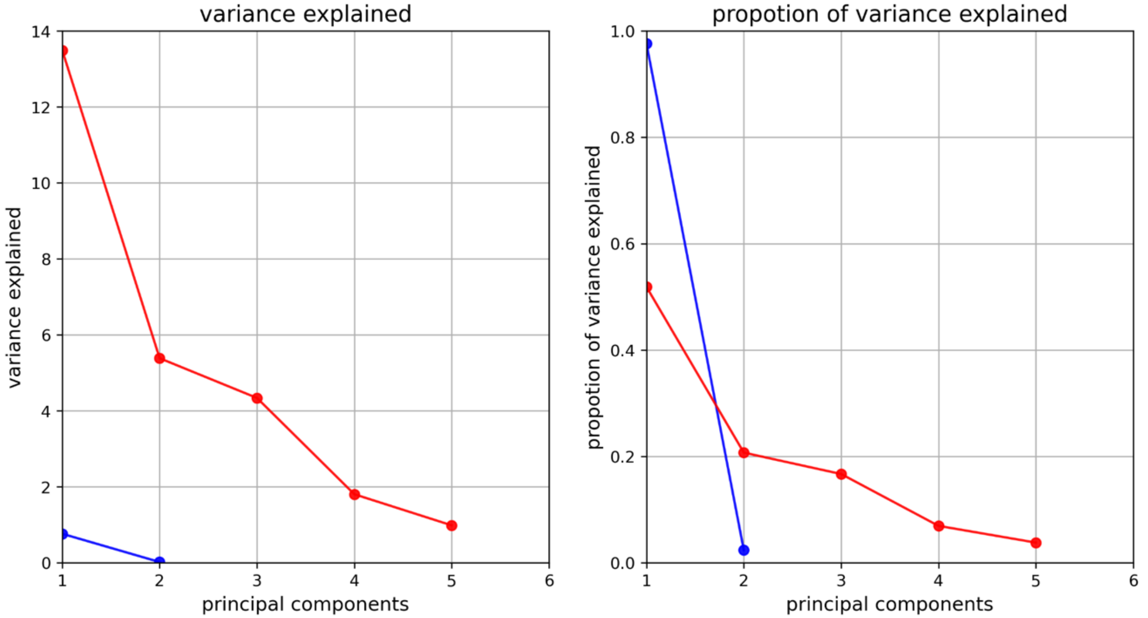
```
[[-2.84777513e-01  1.33828752e-02 -4.01328150e-01  7.56452377e-01
  -4.30625340e-01]
 [ 6.37290518e-01 -6.01836596e-01  2.90984089e-01  3.81084174e-01
  -4.19120474e-02]
 [-4.04144877e-01 -7.88343623e-01 -3.10944893e-01 -3.38567133e-01
  -6.21837647e-02]
 [-4.55042776e-04 -5.10904099e-02 -3.73205245e-01  3.01380387e-01
   8.75943646e-01]
 [-5.91125431e-01 -1.16265701e-01  7.19927145e-01  2.77663127e-01
   2.04110524e-01]]
```

PCA – sklearn.decomposition.pca

```
# eigenvalues
print(pca2D.explained_variance_)
print(pca5D.explained_variance_)
[0.7625315 0.0184779]
[13.48994148  5.38421307  4.33804606  1.80323359  0.98456579]
```

```
# variance explained ratio
print(pca2D.explained_variance_ratio_)
print(pca5D.explained_variance_ratio_)
[0.97634101 0.02365899]
[0.5188439  0.20708512 0.16684793 0.06935514 0.03786791]
```

```
# variance explained ratio
print(pca2D.explained_variance_/sum(pca2D.explained_variance_))
print(pca5D.explained_variance_/sum(pca5D.explained_variance_))
[0.97634101 0.02365899]
[0.5188439  0.20708512 0.16684793 0.06935514 0.03786791]
```



PCA – sklearn.decomposition.pca

```
# project data onto PC1
pca1D = PCA(n_components=1)
pca1D.fit(A2D)
```

$A_{pca} = A_{centered} \times v$

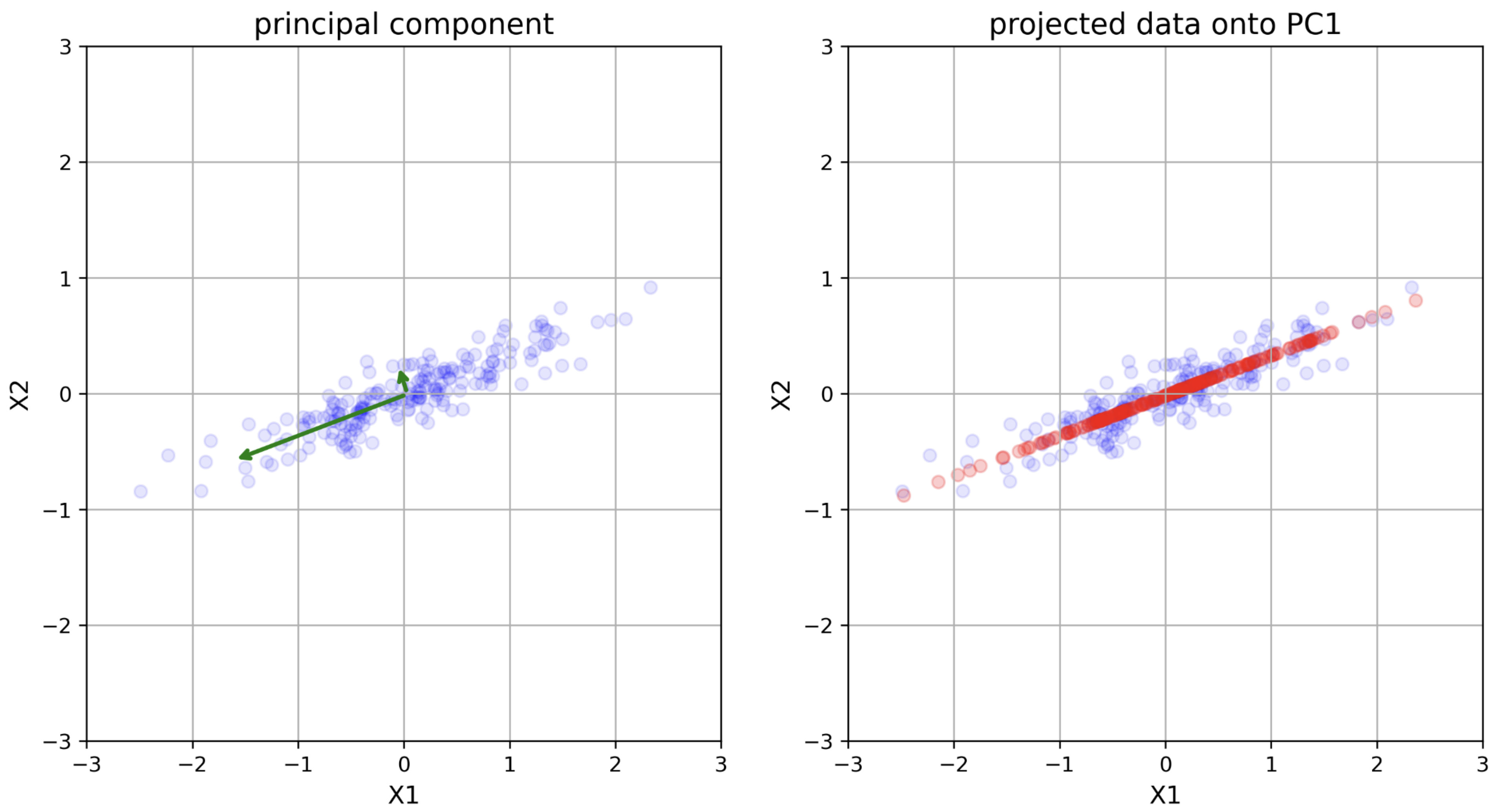
```
A2D_pca1D = pca1D.transform(A2D)
A5D_pca5D = pca5D.transform(A5D)
print("original shape:    ", A2D.shape)
print("transformed shape:", A2D_pca1D.shape)
print("original shape:    ", A5D.shape)
print("transformed shape:", A5D_pca5D.shape)
```

```
original shape:    (200, 2)
transformed shape: (200, 1)
original shape:     (7, 5)
transformed shape: (7, 5)
```

$A_{reconstructed} = A_{pca} \times v^T + \bar{A}$

```
A2D_new1D = pca1D.inverse_transform(A2D_pca1D)
A5D_new5D = pca5D.inverse_transform(A5D_pca5D)
```

PCA – sklearn.decomposition.pca



PCA – sklearn.decomposition.pca

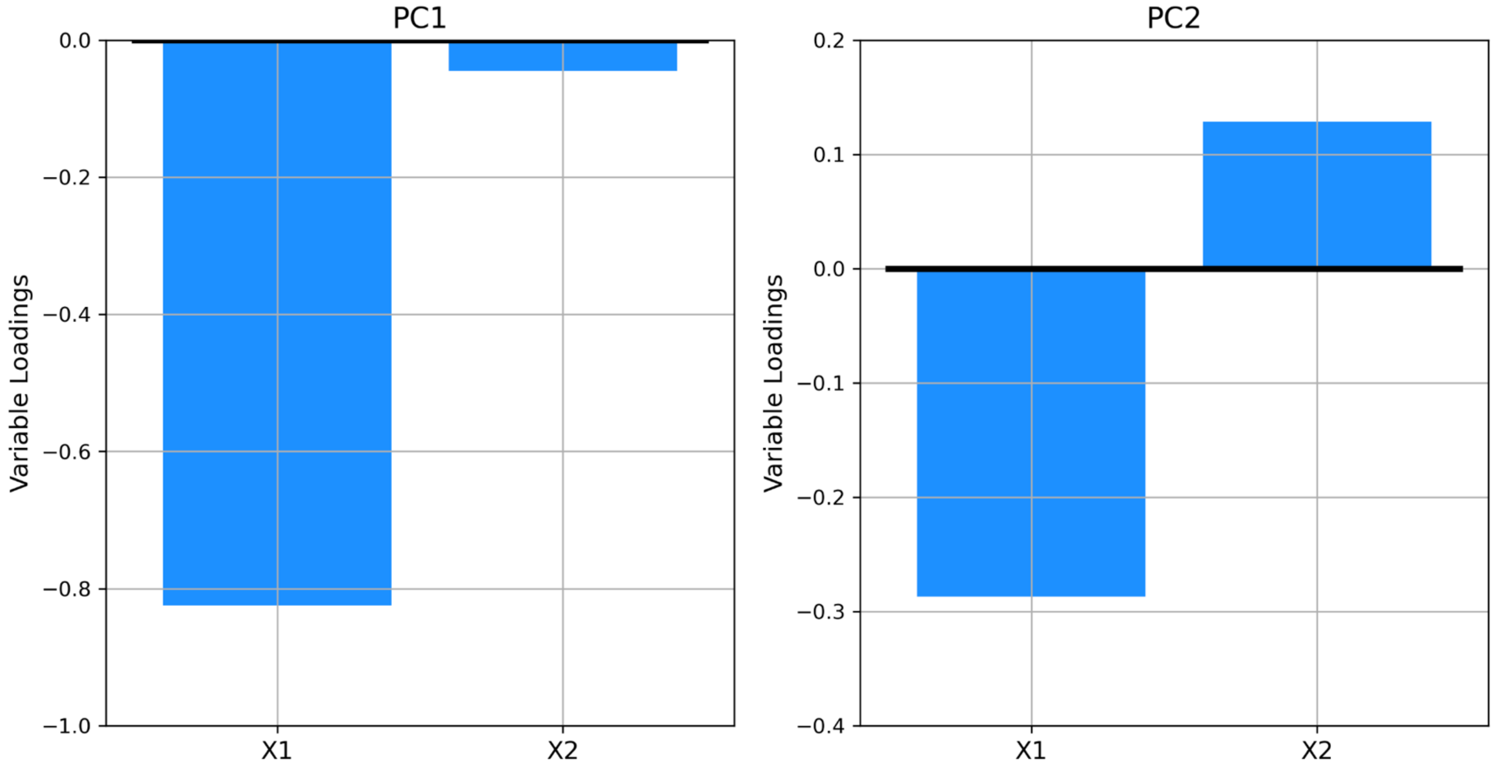
```python
# convert to DataFrame
A5D_new5D = pd.DataFrame.from_records(A5D_pca5D,
        columns=[f"PC{i+1}" for i in range(A5D_new5D.shape[1])])
A5D_new5D
```

	PC1	PC2	PC3	PC4	PC5
0	1.696429	-0.411885	1.282764	2.246521	1.243323
1	-2.084069	1.617905	0.122180	-0.281795	-0.718424
2	3.261464	3.613608	-0.191616	-1.351178	0.764951
3	-4.207169	-2.286135	-2.229844	-0.958474	1.087402
4	5.788023	-2.471383	-1.677120	-0.112444	-0.922710
5	-1.237965	-1.631645	3.888311	-0.961178	-0.436948
6	-3.216714	1.569535	-1.194676	1.418547	-1.017594

PCA – sklearn.decomposition.pca

```python
# variable loading calculation
variable_loading = pca2D.components_.T * np.sqrt(pca2D.explained_variance_)
print(variable_loading)
```

```
[[-0.82473153 -0.0446712 ]
 [-0.28696587  0.12838372]]
```



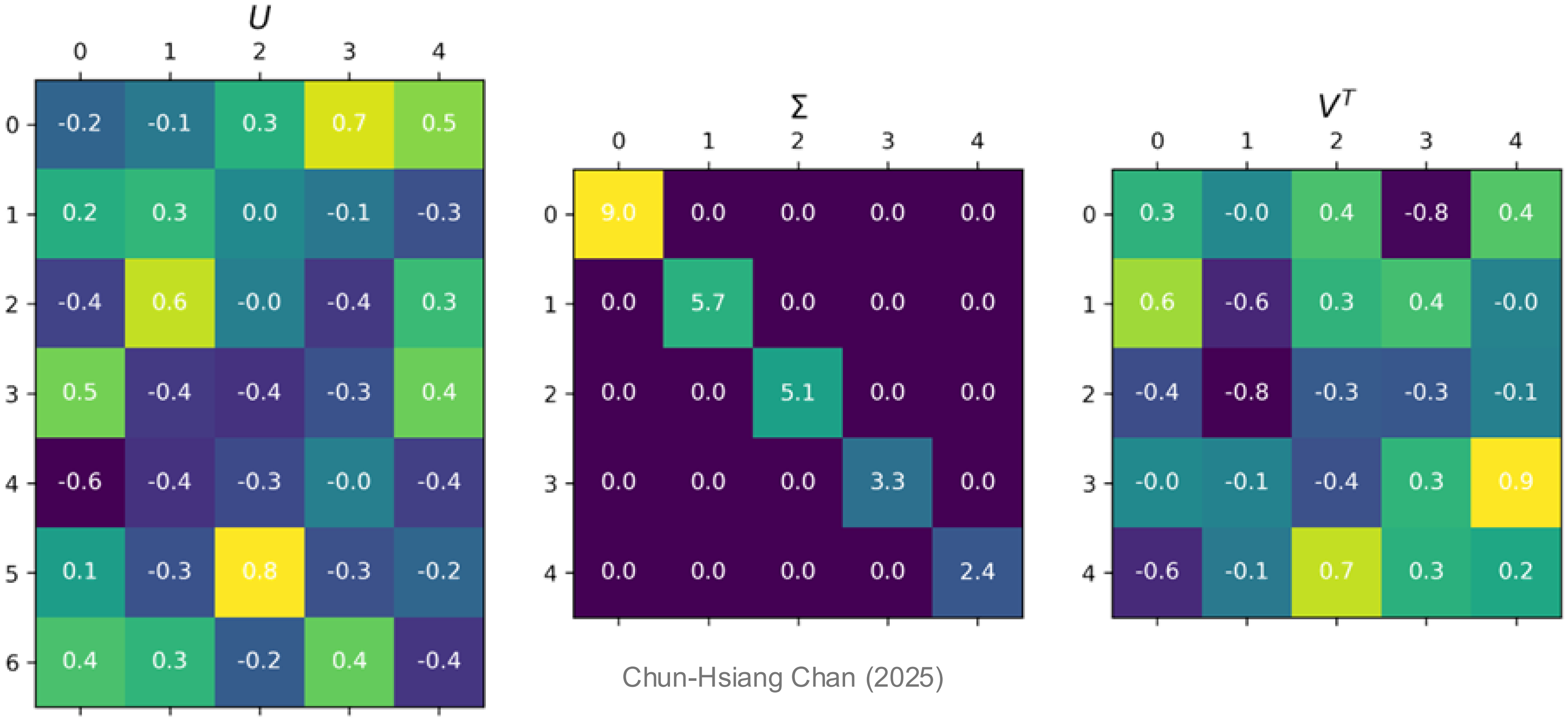
PCA – Singular Value Decomposition

```
# standardized matrix
A_centered = A5D - np.mean(A5D, axis=0)
A_centered
```

```
array([[-2.        , -1.        , -1.14285714,  1.71428571,  1.42857143],
       [ 2.        , -1.        ,  0.85714286, -1.28571429,  0.42857143],
       [ 1.        , -2.        ,  0.85714286,  3.71428571, -2.57142857],
       [ 0.        ,  3.        ,  2.85714286, -3.28571429,  1.42857143],
       [-2.        ,  3.        , -3.14285714,  3.71428571, -2.57142857],
       [-2.        , -2.        , -1.14285714, -3.28571429, -0.57142857],
       [ 3.        ,  0.        ,  0.85714286, -1.28571429,  2.42857143]])
```

PCA – Singular Value Decomposition

```
# compute SVD
U, S, Vt = np.linalg.svd(A_centered, full_matrices=False)
```



PCA – Singular Value Decomposition

```
# proof A = UΣV^T
print(np.round(A_centered, 3))
print(np.round(U @ np.diag(S) @ Vt, 3))
```

```
[[-2.    -1.    -1.143  1.714  1.429]
 [ 2.    -1.     0.857 -1.286  0.429]
 [ 1.    -2.     0.857  3.714 -2.571]
 [ 0.     3.     2.857 -3.286  1.429]
 [-2.     3.    -3.143  3.714 -2.571]
 [-2.    -2.    -1.143 -3.286 -0.571]
 [ 3.     0.     0.857 -1.286  2.429]]
```

```
[[-2.    -1.    -1.143  1.714  1.429]
 [ 2.    -1.     0.857 -1.286  0.429]
 [ 1.    -2.     0.857  3.714 -2.571]
 [ 0.     3.     2.857 -3.286  1.429]
 [-2.     3.    -3.143  3.714 -2.571]
 [-2.    -2.    -1.143 -3.286 -0.571]
 [ 3.     0.     0.857 -1.286  2.429]]
```

PCA – Singular Value Decomposition

```
# extract principal components (PC directions)
principal_components = Vt.T  # Right singular vectors
principal_components
```

```
array([[ 2.84777513e-01,  6.37290518e-01, -4.04144877e-01,
        -4.55042776e-04, -5.91125431e-01],
       [-1.33828752e-02, -6.01836596e-01, -7.88343623e-01,
        -5.10904099e-02, -1.16265701e-01],
       [ 4.01328150e-01,  2.90984089e-01, -3.10944893e-01,
        -3.73205245e-01,  7.19927145e-01],
       [-7.56452377e-01,  3.81084174e-01, -3.38567133e-01,
         3.01380387e-01,  2.77663127e-01],
       [ 4.30625340e-01, -4.19120474e-02, -6.21837647e-02,
         8.75943646e-01,  2.04110524e-01]])
```

PCA – Singular Value Decomposition

```
# compute projected data (Principal Component Scores)
A_pca = U @ np.diag(S)
print(A_pca)
```

```
[[-1.69642934 -0.41188488  1.28276422  2.24652093  1.24332308]
 [ 2.0840688   1.61790489  0.12218009 -0.28179454 -0.71842426]
 [-3.26146374  3.61360798 -0.19161578 -1.35117808  0.76495093]
 [ 4.20716867 -2.28613474 -2.22984393 -0.95847371  1.08740236]
 [-5.78802326 -2.4713829  -1.67711969 -0.11244403 -0.92270986]
 [ 1.23796474 -1.63164506  3.88831104 -0.96117788 -0.4369479 ]
 [ 3.21671412  1.56953472 -1.19467594  1.4185473  -1.01759434]]
```

```
# equivalent to A_centered @ principal_components
A_pca1 = A_centered @ Vt.T
print(A_pca1)
```

```
[[-1.69642934 -0.41188488  1.28276422  2.24652093  1.24332308]
 [ 2.0840688   1.61790489  0.12218009 -0.28179454 -0.71842426]
 [-3.26146374  3.61360798 -0.19161578 -1.35117808  0.76495093]
 [ 4.20716867 -2.28613474 -2.22984393 -0.95847371  1.08740236]
 [-5.78802326 -2.4713829  -1.67711969 -0.11244403 -0.92270986]
 [ 1.23796474 -1.63164506  3.88831104 -0.96117788 -0.4369479 ]
 [ 3.21671412  1.56953472 -1.19467594  1.4185473  -1.01759434]]
```

PCA – Singular Value Decomposition

```
# compute Variance Explained = S2/(N-1)
variance_explained = (S**2) / (A5D.shape[0] - 1)
print(variance_explained)
```

```
[13.48994148  5.38421307  4.33804606  1.80323359  0.98456579]
```

```
# convert to DataFrame for better readability
A_pca_df = pd.DataFrame(A_pca, columns=[f"PC{i+1}" for i
        in range(A_pca.shape[1])])
A_pca_df
```

	PC1	PC2	PC3	PC4	PC5
0	-1.696429	-0.411885	1.282764	2.246521	1.243323
1	2.084069	1.617905	0.122180	-0.281795	-0.718424
2	-3.261464	3.613608	-0.191616	-1.351178	0.764951
3	4.207169	-2.286135	-2.229844	-0.958474	1.087402
4	-5.788023	-2.471383	-1.677120	-0.112444	-0.922710
5	1.237965	-1.631645	3.888311	-0.961178	-0.436948
6	3.216714	1.569535	-1.194676	1.418547	-1.017594

PCA – Eigen Decomposition

centered matrix

A_centered = A5D - np.mean(A5D, axis=0)

A_centered

```
array([[-2.        , -1.        , -1.14285714,  1.71428571,  1.42857143],
       [ 2.        , -1.        ,  0.85714286, -1.28571429,  0.42857143],
       [ 1.        , -2.        ,  0.85714286,  3.71428571, -2.57142857],
       [ 0.        ,  3.        ,  2.85714286, -3.28571429,  1.42857143],
       [-2.        ,  3.        , -3.14285714,  3.71428571, -2.57142857],
       [-2.        , -2.        , -1.14285714, -3.28571429, -0.57142857],
       [ 3.        ,  0.        ,  0.85714286, -1.28571429,  2.42857143]])
```

compute the covariance Matrix

S = np.cov(A_centered, rowvar=False)

S

```
array([[ 4.33333333e+00, -6.66666667e-01,  2.66666667e+00,
        -1.16666667e+00,  1.50000000e+00],
       [-6.66666667e-01,  4.66666667e+00,  7.40148683e-17,
        -1.48029737e-16,  1.66666667e-01],
       [ 2.66666667e+00,  7.40148683e-17,  3.80952381e+00,
        -3.04761905e+00,  1.90476190e+00],
       [-1.16666667e+00, -1.48029737e-16, -3.04761905e+00,
         9.23809524e+00, -3.85714286e+00],
       [ 1.50000000e+00,  1.66666667e-01,  1.90476190e+00,
        -3.85714286e+00,  3.95238095e+00]])
```

PCA – Eigen Decomposition

```
# compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(S)
print(eigenvalues)
[13.48994148  5.38421307  4.33804606  0.98456579  1.80323359]
print(eigenvectors)
[[-2.84777513e-01  6.37290518e-01 -4.04144877e-01  5.91125431e-01
  -4.55042776e-04]
 [ 1.33828752e-02 -6.01836596e-01 -7.88343623e-01  1.16265701e-01
  -5.10904099e-02]
 [-4.01328150e-01  2.90984089e-01 -3.10944893e-01 -7.19927145e-01
  -3.73205245e-01]
 [ 7.56452377e-01  3.81084174e-01 -3.38567133e-01 -2.77663127e-01
   3.01380387e-01]
 [-4.30625340e-01 -4.19120474e-02 -6.21837647e-02 -2.04110524e-01
   8.75943646e-01]]
```

PCA – Eigen Decomposition

```
# sort eigenvectors by largest eigenvalues
sorted_indices = np.argsort(eigenvalues)[::-1]  # descending order
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
print(sorted_indices)
print(eigenvalues)
print(eigenvectors)
```

```
[0 1 2 4 3]
```

```
[13.48994148  5.38421307  4.33804606  1.80323359  0.98456579]
```

```
[[-2.84777513e-01  6.37290518e-01 -4.04144877e-01 -4.55042776e-04
   5.91125431e-01]
 [ 1.33828752e-02 -6.01836596e-01 -7.88343623e-01 -5.10904099e-02
   1.16265701e-01]
 [-4.01328150e-01  2.90984089e-01 -3.10944893e-01 -3.73205245e-01
  -7.19927145e-01]
 [ 7.56452377e-01  3.81084174e-01 -3.38567133e-01  3.01380387e-01
  -2.77663127e-01]
 [-4.30625340e-01 -4.19120474e-02 -6.21837647e-02  8.75943646e-01
  -2.04110524e-01]]
```

PCA – Eigen Decomposition

```
# compute principal component scores (project data)
# transform data onto PCA space
A_pca = A_centered @ eigenvectors
print(A_pca)
```

```
[[ 1.69642934 -0.41188488  1.28276422  2.24652093 -1.24332308]
 [-2.0840688   1.61790489  0.12218009 -0.28179454  0.71842426]
 [ 3.26146374  3.61360798 -0.19161578 -1.35117808 -0.76495093]
 [-4.20716867 -2.28613474 -2.22984393 -0.95847371 -1.08740236]
 [ 5.78802326 -2.4713829  -1.67711969 -0.11244403  0.92270986]
 [-1.23796474 -1.63164506  3.88831104 -0.96117788  0.4369479 ]
 [-3.21671412  1.56953472 -1.19467594  1.4185473   1.01759434]]
```

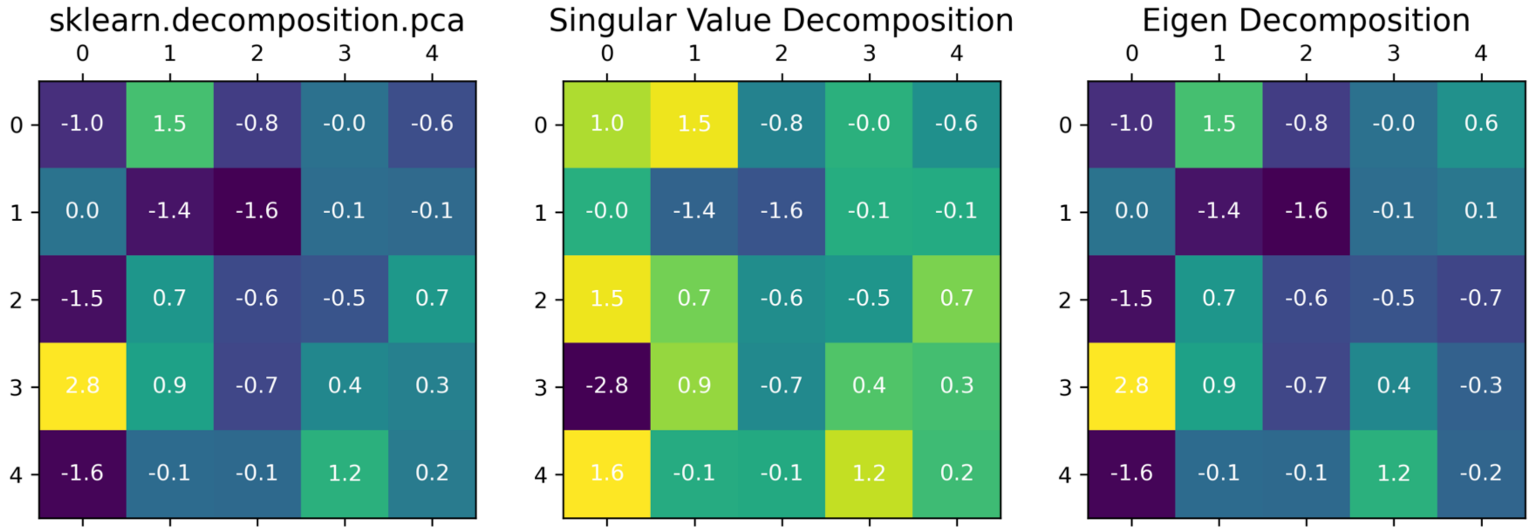
PCA – Eigen Decomposition

```
# convert to DataFrame for better readability
A_pca_df = pd.DataFrame(A_pca, columns=[f"PC{i+1}" for i
        in range(A_pca.shape[1])])
A_pca_df
```

	PC1	PC2	PC3	PC4	PC5
0	1.696429	-0.411885	1.282764	2.246521	-1.243323
1	-2.084069	1.617905	0.122180	-0.281795	0.718424
2	3.261464	3.613608	-0.191616	-1.351178	-0.764951
3	-4.207169	-2.286135	-2.229844	-0.958474	-1.087402
4	5.788023	-2.471383	-1.677120	-0.112444	0.922710
5	-1.237965	-1.631645	3.888311	-0.961178	0.436948
6	-3.216714	1.569535	-1.194676	1.418547	1.017594

PCA – Variable Loading

```
# PCA :: variable loading
loading = pca5D.components_.T * np.sqrt(pca5D.explained_variance_)
# SVD :: variable loading
loading1 = principal_components * np.sqrt(variance_explained)
# Eigen :: variable loading
loading2 = eigenvectors * np.sqrt(eigenvalues)
```



The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw
Website: https://toodou.github.io/

STDS Lab
Est. 2022